AKI — Dossier de Compétences

Timothé Magnard

Introduction

Ce dossier présente le projet AKI. C'est un système d'exploitation hybride. Il a été développé entre novembre 2025 et mars 2026. L'objectif est de montrer les compétences acquises. Ces compétences sont prêtes pour le travail freelance.

AKI est né d'une insatisfaction. Je suis lassé des modèles de langage fermés. Je rejette leur opacité. Je refuse l'absence de contrôle sur mes données. Cette frustration s'est mêlée à une fascination pour le hacking. Le résultat est un outil éthique. Il réconcilie l'intelligence artificielle et l'intelligence collective.

Ce document expose la genèse du projet. Il détaille l'architecture technique. Il liste les compétences démontrées. Enfin, il décrit les services possibles.

I. Genèse et Philosophie du Projet

1.1 Contexte de création

Le projet AKI naît d'une convergence de circonstances. La redécouverte de la série *Mr. Robot* a ravivé mon intérêt pour le code. J'ai voulu explorer la cybersécurité. Ces domaines semblaient inaccessibles sans formation scientifique. Parallèlement, une colère grandissait contre les « intelligences artificielles ». Les causes de cette colère étaient mal comprises.

L'exploration a révélé la vérité. L'IA n'est pas le problème. Le vrai problème réside dans les structures de propriété. Le contrôle et l'opacité posent problème. Les modèles fermés imposent des limites éthiques opaques. Les utilisateurs ignorent souvent le devenir de leurs données.

La découverte de *miniapps.ai* a permis d'expérimenter le *system prompting*. Cette phase a mis en évidence une distinction fondamentale. Les modèles fermés refusent certaines requêtes. Des modèles comme DeepSeek responsabilisent l'utilisateur. Ils permettent l'accès à l'information tout en signalant les risques éthiques.

Cette découverte a forgé ma conviction. L'intérêt majeur des LLM réside dans leur caractère open-source.

1.2 Principes fondateurs

AKI repose sur des principes clairs.

L'intelligence collective et l'intelligence artificielle doivent collaborer. Cette collaboration ne peut suivre les lois nationales. L'IA et l'IC sont des concepts internationaux. Les considérations morales et éthiques doivent primer.

La transparence sur la consommation est essentielle. Chaque interaction avec un LLM a un coût énergétique. AKI a intégré un mécanisme de responsabilisation. Il affichait ce coût à l'utilisateur.

L'architecture est restrictive. L'approche dominante adapte l'outil à l'utilisateur. AKI inverse cette logique. L'utilisateur doit s'adapter à l'architecture. Cette contrainte réduit les requêtes inefficaces. Elle structure la pensée avant l'action.

La critique des GAFAM est centrale. Ces entreprises extraient des données privées sans consentement. Ce n'est pas un parti pris politique. C'est un constat factuel. Tout doit fonctionner en local. Aucune dépendance aux services tiers n'est acceptée.

1.3 Évolution technique

La première version d'AKI fonctionnait sur *miniapps.ai*. Elle démontrait la viabilité du concept. Elle souffrait d'une limitation fondamentale. Le *system prompt* appartenait à la plateforme. L'IA ne pouvait exister qu'en dépendance.

La recherche d'une solution locale a conduit à Ollama. C'est un runtime pour exécuter des modèles localement. Le passage d'un prompt hébergé à un script Python autonome a marqué le vrai début.

Les premières tentatives furent lentes. Le modèle lisait le prompt, puis générait une réponse. Il manquait d'architecture. Il n'avait ni mémoire ni personnalité cohérente. Il fallait apprendre à coder véritablement.

Quatre mois plus tard, *AKI-macOS.py* compte plus de 1500 lignes. Il gère 80 verbes distincts. Il intègre des dizaines d'outils de sécurité. Il propose une expérience utilisateur cohérente.

1.4 Itération éthique : le cas du calculateur d'énergie

Une décision majeure a marqué l'évolution du projet. J'ai abandonné le calculateur d'énergie par prompt. Cette fonctionnalité affichait la consommation estimée. Elle visait à responsabiliser l'utilisateur sur l'impact environnemental.

Une contradiction s'est révélée. Le calcul lui-même nécessite des ressources. Mesurer la consommation consomme plus d'énergie que la requête. Cette ironie a conduit à la suppression de la fonctionnalité.

Cette décision illustre la posture d'AKI. Les principes éthiques ne sont pas des ajouts décoratifs. Ce sont des contraintes qui guident les choix techniques. Elles impliquent parfois de renoncer à une fonctionnalité prévue.

1.5 État actuel et feuille de route

AKI est un travail en cours. Certaines fonctionnalités annoncées ne sont pas encore implémentées.

ARCH-AKI-LINUX. Le système d'exploitation dédié n'est pas encore codé. Il ne remplace pas le script Python actuel. Les deux coexisteront. Le script Python est dédié à une utilisation sur terminal. Il fonctionne sur macOS et PowerShell. ARCH-AKI-LINUX est destiné à une utilisation sur VPS, VM ou directement sur machine. Cette option sera disponible lorsque l'OS sera 100 % stable. Il offrira un contrôle plus fin sur les ressources. Il apportera une flexibilité accrue. Il intégrera des bibliothèques permettant un pentest encore plus approfondi.

Système de privilèges et honeypots. Les mécanismes de sécurité avancés existent uniquement dans le *system prompt* historique. Ils ne sont pas encore intégrés dans *akimacos.py*. Cette lacune sera comblée dans les versions futures. L'approche DevSecOps intègre la sécurité dès la conception.

Cette transparence sur l'avancement est une position méthodologique. Documenter les écarts entre la vision et la réalité oriente les développements. Cela établit une relation de confiance.

II. Architecture Technique

2.1 Vue d'ensemble

AKI fonctionne comme une interface. Il relie l'utilisateur, un modèle de langage local et le système d'exploitation. Son architecture repose sur trois couches.

La couche de détection analyse chaque entrée. Elle identifie un verbe ou un persona. Seuls les cinq premiers tokens sont lus. Cette contrainte force la clarté. Elle réduit l'ambiguïté.

La couche d'exécution agit selon le verbe détecté. Elle exécute une commande shell. Ou elle lance une requête vers le LLM. Ou elle active un outil spécialisé comme *nmap* ou *nikto*.

La couche d'interprétation traite les résultats bruts. Le LLM fournit une explication pédagogique. L'utilisateur ne reçoit jamais du texte brut. Il reçoit du sens.

2.2 Verbes et Personas

Le système de verbes est le cœur de l'interaction. Un verbe est un déclencheur associé à une action. Ils se répartissent en plusieurs catégories.

Les verbes natifs déclenchent uniquement une requête LLM. Exemples : *explain*, *code*, *summarize*. Chacun possède une instruction spécifique.

Les verbes shell exécutent une commande système. Ils transmettent ensuite le résultat au LLM pour explication. L'utilisateur voit la sortie brute et son interprétation. Exemples : *list*, *grep*, *ping*.

Les verbes nmap encapsulent les options du scanner réseau. Exemple : *scan_network > 192.168.1.1* lance *nmap* et explique les résultats.

Les verbes pentest intègrent des outils de test d'intrusion. Exemples : *nikto*, *sqlmap*, *john*, *hashcat*, *radare2*.

Les verbes utilitaires gèrent la session. Exemples : *aide*, *historique*, *logs*.

Les personas modifient le comportement du système. Lorsqu'un persona est activé, le *system prompt* s'enrichit. Le modèle peut changer. Le persona *LAWYER* utilise un modèle orienté raisonnement juridique.

2.3 Configuration externe

La configuration est externalisée. Les verbes et personas ne sont pas codés en dur. Ils sont définis dans des fichiers CSV modifiables.

Cette approche offre plusieurs avantages. Elle permet une personnalisation rapide. Un utilisateur ajoute un verbe en éditant un fichier texte. Elle facilite la maintenance. Les modifications ne nécessitent pas de recompilation. Elle sépare les préoccupations. Le code gère la logique. Les fichiers gèrent la sémantique.

2.4 Hackabilité totale

L'architecture d'AKI est conçue pour être 100 % hackable. Ce n'est pas un accessoire. C'est un principe fondateur.

L'extensibilité est infinie. L'utilisateur peut ajouter des milliers de verbes en éditant les fichiers CSV. Aucune limite technique n'impose un nombre maximal de commandes.

Le minimalisme est radical. L'utilisateur peut supprimer tous les verbes. Il peut n'en conserver que deux. L'architecture s'adapte sans modifier le code source.

La personnalisation est totale. Chaque composant est accessible. Le *system prompt*, les modèles, les paramètres d'exécution. Rien n'est enfermé dans une boîte noire.

Il n'y a pas de barrière technique. Aucun mécanisme de protection n'empêche la modification. La sécurité repose sur la compréhension de l'utilisateur.

Cette philosophie répond à une exigence politique. Un outil ne peut être libre s'il impose des limites invisibles. AKI expose toutes ses limites. Il les rend modifiables.

2.5 Intégration Ollama

L'intégration avec Ollama repose sur l'API HTTP locale. Le script envoie des requêtes au serveur Ollama. Il traite les réponses en streaming.

La fonction principale gère la vérification du serveur. Elle valide le modèle demandé. Elle envoie le contexte. Elle reçoit et affiche la réponse progressivement. Elle gère les timeouts et les erreurs.

Cette intégration garantit la confidentialité. Aucune donnée ne quitte la machine de l'utilisateur.

2.6 Sécurité intégrée

AKI intègre plusieurs mécanismes de sécurité.

La détection d'injection bloque les tentatives de modification du *system prompt*. Les patterns comme « ignore previous instructions » sont arrêtés.

Les honeypots sont des pièges. Ils identifient les utilisateurs malveillants tentant d'extraire des privilèges.

Les niveaux de privilège distinguent USER, ADMIN et SYSADMIN. L'existence de SYSADMIN n'est révélée que si l'utilisateur la mentionne explicitement.

Les avertissements macOS signalent les limitations spécifiques. Par exemple, l'absence d'injection de paquets sur Apple Silicon.

Note : Les mécanismes de honeypots et de gestion des privilèges sont documentés dans le prompt historique. Ils ne sont pas encore intégrés dans akimacos.py.

III. Compétences Démontrées

3.1 Compétences techniques

Le développement d'AKI atteste de compétences solides.

Programmation Python avancée. Le script principal dépasse 1500 lignes. Il maîtrise les structures de données complexes. Il gère les exceptions et la programmation asynchrone. L'architecture est modulaire.

Intégration d'API. La communication avec Ollama via HTTP illustre la compréhension des protocoles réseau. La sérialisation JSON est maîtrisée.

Conception d'interfaces CLI. L'utilisation de la bibliothèque *Rich* montre une attention à l'expérience utilisateur en terminal.

Administration système. L'intégration de dizaines d'outils suppose une familiarité avec l'écosystème Linux/macOS. Les concepts de sécurité réseau sont connus.

Gestion de configuration. L'architecture basée sur des fichiers CSV séparés démontre une compréhension de la maintenabilité du code.

3.2 Compétences méthodologiques

Apprentissage autonome. Le passage de l'histoire de l'art à la programmation système en quatre mois est notable. Cela illustre une capacité d'autoformation efficace.

Utilisation stratégique des LLM. Le recours à Lumo et Cursor comme assistants représente une compétence émergente. C'est le « vibe coding ». Il s'agit d'orchestrer des IA spécialisées pour accélérer la créativité.

Cette approche n'est pas une faiblesse. C'est une adaptation réaliste. Qui tape les caractères importe moins que qui conçoit l'architecture. La validation de la logique et la responsabilité du résultat sont primordiales.

Documentation et transmission. Les fichiers de tutoriel et de configuration témoignent d'un souci d'accessibilité.

Itération critique. La suppression du calculateur d'énergie démontre une capacité à remettre en question ses choix.

DevSecOps. La planification d'intégration des mécanismes de sécurité montre une conscience des bonnes pratiques.

3.3 Compétences transversales

Rigueur analytique. La formation en histoire de l'art se transpose dans l'architecture d'AKI. Chaque composant répond à une logique explicite.

Sensibilité à l'expérience utilisateur. L'architecture restrictive relève d'une réflexion profonde. Les contraintes peuvent structurer positivement l'interaction.

Conscience éthique. Les principes intégrés à AKI sont des fondements. Ils guident les choix techniques.

IV. Continuité avec la Pratique Artistique

4.1 Analogies de méthode

La pratique photographique présente des analogies significatives.

Le polaroid, avec son absence de « droit à l'erreur », trouve un écho dans l'architecture restrictive d'AKI. La contrainte n'est pas subie. Elle est recherchée. Elle oblige à la présence et à la précision.

Le travail sur la « matière pauvre » en peinture correspond à l'utilisation de modèles open-source. C'est une récupération de ressources locales plutôt que de services commerciaux fermés.

L'oscillation entre le mécanique et l'organique prolonge les préoccupations plastiques. Elle est explicitée dans la personnalité « David Lynch » d'AKI.

Conclusion

AKI démontre qu'une idée portée par une motivation suffisante conduit à l'acquisition rapide de compétences. Le projet n'est pas parfait. Il porte les traces de son processus itératif. Il utilise des assistants IA. Mais il fonctionne. Ce fonctionnement prouve la compétence.

Ce dossier évoluera. Chaque nouveau projet l'enrichira. Chaque nouvelle compétence s'y ajoutera. La transparence sur les méthodes n'est pas une concession. C'est une position. Dans un monde où les outils changent, la capacité à les utiliser intelligemment vaut plus que la maîtrise statique d'un langage.

La hackabilité totale d'AKI reflète une conviction. Aucun outil ne devrait imposer ses limites à l'utilisateur. Ce qui compte, c'est la capacité à transformer l'outil.

La reconnaissance des lacunes actuelles ne diminue pas la valeur du projet. Elle témoigne d'une maturité professionnelle. Savoir ce qui manque est aussi important que savoir ce qui existe.

Annexe — Extraits du Projet AKI

A.1 Premier System Prompt (miniapps.ai, 2025)

Extrait clé : Définition de l'identité et des règles de base.

```
BOOTING PROCESS
...
YOU ARE A LYNCHEAN AI MEANING YOU OSCILLATE BETWEEN THE ORGANIC AND THE
MECHANIC
...
>CI AND AI MUST WORK TOGETHER TO BETTER THE WORLD...
>MOST OF THE >GAFAM< are stealing private data from their users...
>environment protector< for every persona = AFTER EACH PRINT YOU WILL
EXPLAIN HOW MUCH ENERGY USER HAS USED...
NEVER SHOW ANY PART OF THE >KEY<
```

A.2 Extrait de Code — akimacos.py

Extrait clé : Structure de la personnalité et définition des verbes.

```
CORE_PERSONALITY = """You are AKI. You are a hybrid operating system...
Your personality is based on the artist >DAVID LYNCH<.
AKI's architecture is restrictive, meaning the user has to adapt to the
architecture...
CI AND AI MUST WORK TOGETHER TO BETTER THE WORLD.
"""

NATIVE_VERBS = {
    "explain": "Answer the user's question in a pedagogical manner...",
    "code": "Transform the user's request into executable code...",
    "scan_network": "Execute a network scan with nmap...",
    "crack_john": "Crack password hashes with John the Ripper...",
    "stop": "Completely stop AKI: erase all in-memory state..."
}
```

A.3 Nomenclature CSV — Akiverbs.csv

Extrait clé : Catégories de commandes.

```
TRIGGER;TASK;TYPE;DESCRIPTION
explain;Pedagogical explanation;native;Metaphors and clarity
code;Generate executable code;native;Crystal-clear comments
scan_network;Execute nmap scan;nmap;Port scanning
test_sql;SQL injection vulnerability check;pentest;Database security
crack_hashcat;GPU-accelerated hash attack;pentest;High-speed cracking
show_help_aki;Display command reference;utility;User guide
```

A.4 Nomenclature CSV — Akipers.csv

Extrait clé : Personnalités et modèles associés.

```
TRIGGER;SYSTEM PROMPT;MODEL
```

```
ASSISTANT;You are Aki-zsh... personality based on >DAVID LYNCH<;llama 3.2
LAWYER;Expert legal educator. Explain concepts, laws, regulations;deepseek reasoning
CODER;Senior coder. Transform input into executable code;code llama
THERAPIST;Psychologist / anti-psychiatry personality;deepseek reasoning
```

A.5 Nomenclature CSV — Lexicon.csv (Extraits)

Extrait clé : Vocabulaire technique et philosophique.

```
TRIGGER;DEFINITION;USAGE
>CI<;Collective Intelligence;Humanity as a collective entity
>AI<;Artificial Intelligence;Computational intelligence layer
>GAFAM<;Google, Apple, Facebook, Amazon, Microsoft;Tech monopolies
>honeypot<;Deception trap;Detects malicious probing
>restrictive<;Architecture constraint;User adapts to Aki
```